Using dialogue to learn math in the LeActiveMath project

Using Dialogue to Learn Math in the LeActiveMath Project

Charles Callaway and **Myroslava Dzikovska** and **Colin Matheson** and **Johanna Moore**[1] and **Claus Zinn**[2]

Abstract.

We describe a tutorial dialogue system under development that assists students in learning how to differentiate equations. The system uses deep natural language understanding and generation to both interpret students' utterances and automatically generate a response that is both mathematically correct and adapted pedagogically and linguistically to the local dialogue context. A domain reasoner provides the necessary knowledge about how students should approach math problems as well as their (in)correctness, while a dialogue manager directs pedagogical strategies and keeps track of what needs to be done to keep the dialogue moving along.

1 Overview

One-on-one tutoring is known to be better in helping students learn when compared with reading textbooks. Human tutors typically produce effect sizes of up to two standard deviations [2] compared to unsupervised reading. Tutorial systems, in particular cognitive tutors which model the inner state of a student's knowledge, improve over reading alone but still result in only up to 1 standard deviation effect size [1]. One potential reason for this difference is the use of interactive dialogue, which allows students to freely ask questions and tutors to adapt their direct feedback and presentation style to the individual student's needs.

Adding natural language dialogue to a tutorial system is a complex task. Many existing tutorial dialogue systems rely on pre-authored curriculum scripts [15] or finite-state machines [16] without detailed knowledge representations. These systems are easy to design for curriculum providers, but offer limited flexibility because the writer has to predict all possible student questions and answers. Representations of domain knowledge and reasoning, along with a record of past student mistakes and misconceptions, is vital for adaptively interacting with students via natural language.

We describe a tutorial system for solving differential equations which uses a custom-built domain reasoner, named SLOPERT, capable of determining whether a student's answer is correct or not, and if not, what the most likely explanation for the mistake or misconception would be. Our approach relies on using an existing wide-coverage parser for domain-independent syntactic parsing and semantic interpretation, a dialogue manager (DM) using the information-update approach [14], the SLOPERT domain reasoner for the domain of differential equations, and a wide-coverage deep generation system for adaptively generating tutor utterances. We hope to show that despite the significant effort involved in representing knowledge and integrating it with deep natural language components, students will learn more because the dialogue they engage in will be more similar to that with a human tutor.

The tutorial dialogue system presented here is one component of the LeActiveMath project, a sixth European Framework project for developing eLearning systems for high school and college or university level classrooms which can be used in informal contexts for self learning. The core system provides lessons customized to the learner by combining hand-written text and exercises, adapts to the learner and learning context, and also contains functionality for open learner modeling, the personalization of the lesson material, and interactivity for active and exploratory learning. Our tutorial dialogue system receives requests to help a student solve particular problems and reports any mistakes (in the form of misconceptions diagnosed) to the learner model.

A typical interaction with the tutorial dialogue system consists of solving a series of problems proposed by either tutor or student, centering around differentiation based on the chain rule. The student proceeds to solve each problem one by one, and can directly ask the tutor for help or receive help proactively when the tutor detects errors in the student's intermediate steps. The use of a domain reasoner allows us to know exactly what errors a student is likely making at any point in time, which can be used as a basis for adaptive feedback by the generation system. The types of feedback available and methods for selecting what to say at any stage are informed by a corpus of 19 collected human-human tutorial interactions [7].

2 Dialogue System Architecture

A student interacting with the prototype of our intelligent tutoring system for differential equations sees the graphical user interface shown in Fig. 1. It consists of a mixed text and math formula entry area, a set of buttons indicating the purpose of the student's input, the current problem under discussion, a large area for holding the history of the dialogue up to this point (both tutor's and student's utterances), and an area for holding the previous term entered as a cut-and-paste convenience for students. The two text fields allow students to type text both before and after a mathematical formula (entering only a mathematical formula with no text is usually indicative of an intermediate substep or the final solution).

The architecture of the tutorial dialogue system with respect to the overall LeActiveMath system is shown in Fig. 2. There are three main parts: (i) the server-side dialogue system described here, (ii) a client-side Java-based GUI applet supporting student interaction (Fig. 1) called G(LE)^{2}AM, and (iii) the LeActiveMath core system described above. Communication between G(LE)^{2}AM and the dialogue manager (DM) as well as between the DM and the LeActive-

[1] University of Edinburgh, HCRC email: ccallawa@inf.ed.ac.uk, {m.dzikovska,Colin.Matheson,J.Moore}@ed.ac.uk

[2] DFKI, zinn@dfki.de

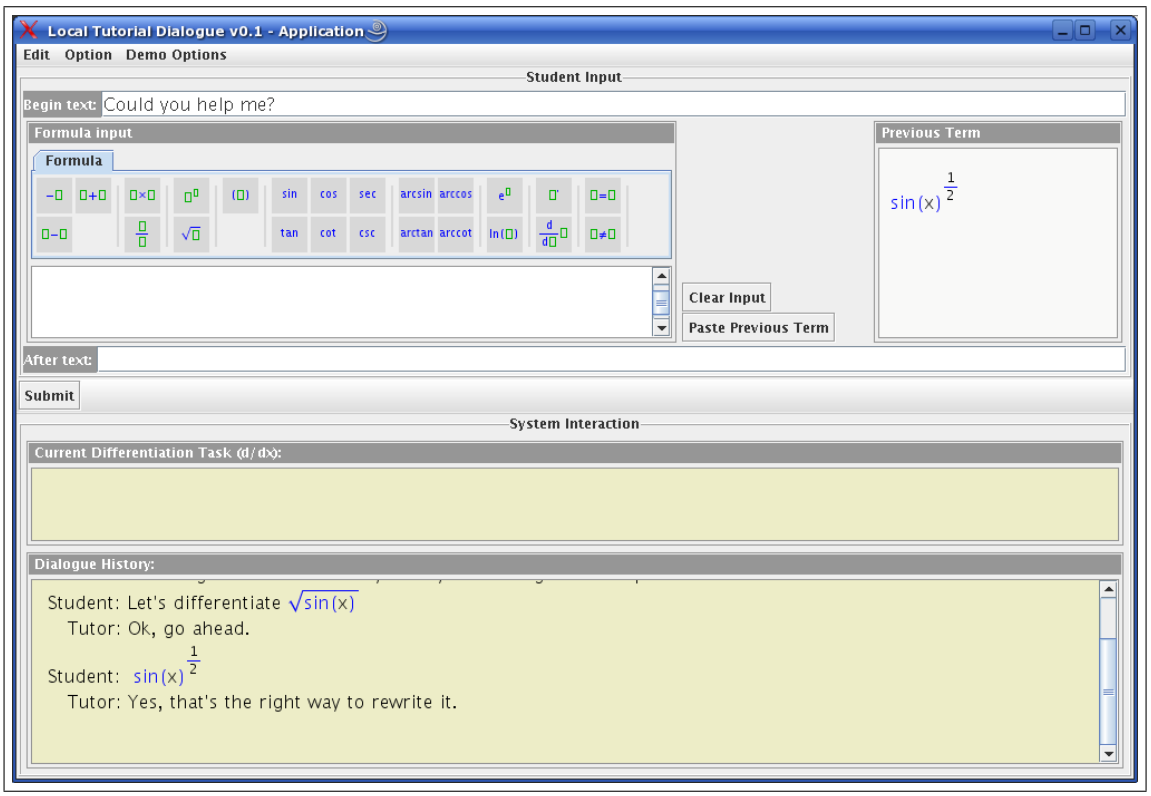

Figure 1. The G(LE)^{2}AM user interface for the tutorial dialogue component.

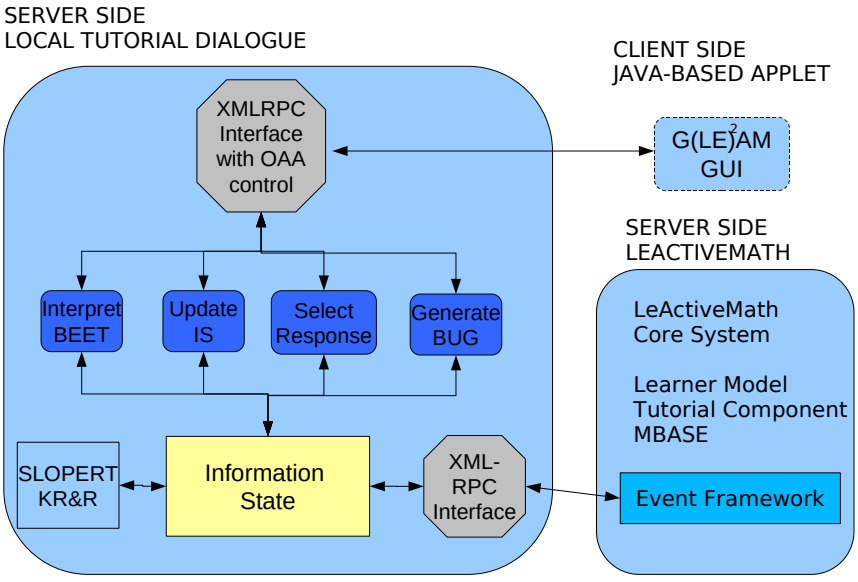

Figure 2. Architecture of the tutorial dialogue component in relation to other LeActiveMath components.

Math core system is established via XML-RPC. Problems to solve can be proposed by either the student or the LeActiveMath core system. Information about students' capabilities in problem solving is exchanged with the learner modeling component of LeActiveMath to allow adaptive advice to be given for both the current and future math problems.

3 Natural Language Understanding

Understanding the sentences typed in by the student places a number of requirements on the NLU components involved in tutorial dialogue: dealing with complex and varied syntactic constructs produced by the students, as well as fragmentary utterances; parsing mathematical expressions interleaved with English; determining what is in the common ground, and particularly interpreting referring expressions; and handling referring expressions with math elements.

Our BEET natural language interpretation component consists of:

- The TRIPS parser and grammar [8] with extensions to handle interleaved natural language and mathematical expressions;
- A spelling corrector to deal with typed input;
- A dialogue move identification algorithm, currently capable of identifying student help requests and statements of confusion.

The TRIPS parser is a bottom-up chart parser using a feature-based unification grammar formalism It comes with a wide-coverage grammar that is geared specifically for dialogue interpretation and a domain-independent lexicon, and produces semantic interpretations in parallel with syntactic analysis. The TRIPS grammar contains 381 grammar rules which provide wide coverage of various syntactic constructs. In addition to handling complex syntactic phenomena, TRIPS provides good coverage for fragments found in spoken dialogue.

BEET also contains an initial implementation of a reference resolution algorithm capable of identifying expressions to resolve, and querying the reasoners for the necessary information.

3.1 Parsing Interleaved Symbolic and Natural Language

The design of the multimodal interface (Figure 1) gives the student the option to enter math expressions in the formula editor window together with free natural language input. While lack of accompanying text often indicates a substep or final solution to the problem, there are many possible motives for the students to add text: explaining their reasoning or knowledge of the domain to the tutor, asking for help, editing previous problem steps, asking clarification questions, and hedging when proposing the next substep to name a few.

Our corpus contains 19 human-human dialogues on solving differentiation problems for a total of 1010 turns and 53.16 turns per dialogue. 54% of student turns consisted of only a term or formula entered in the equation editor window. BEET does not perform analysis of the mathematical input; it marks this language-free input as a potential contribution to the current problem solution, and delegates its analysis to the domain reasoner, which decides whether the student's math expression fits the given task context.

When students chose to use the free text field for input, in 63% of the cases they did not use any mathematical expressions, but in 36% of cases the student contribution contained both text and mathematical terms. If the student entered both language and mathematical expressions, they were frequently integrated tightly:

```
(a) (Speechact V1 WH-Question :Content E2)
      (F E2 (:* LF::Happen happen) :Patient V3 :Theme V4
            :TMA ((Tense pres)
                  (Modality (:* LF::Conditional would))))
            (THE V3 LF::Term :refers-to-external term295)
            (WH-TERM V4 (:* LF::Referential-sem what)
                  :Context-rel what)

(b) <OMOBJ id="term295">
      <OMS cd="arith1" name="power"/>
         <OMV name="x"/>
         <OMI>3</OMI>
      </OMS></OMOBJ>
```

Figure 3. The representation for *What would happen to the* x^3: (a) the TRIPS LF representation using `refers-to-external` to refer to math input; (b) the OpenMath representation for the math term x^3.

(1) What would happen to the x^3
(2) I know that $sin'(x) = cos(x)$

Here, the mathematical expressions serve as parts of utterances (such as noun or verb phrases), which would not be complete without them. Thus the interpretater has to handle interleaved language and mathematical expressions, which presents a technical problem.

The TRIPS domain-independent representation necessary to represent natural language content is a flattened semantic representation using semantic types and roles and is not suitable for representing the details of mathematical expressions. In contrast, OpenMath (`http://www.openmath.org`) is an XML format built specifically for representing mathematical expressions, but which offers no support for representing natural language.

In BEET, we tackle this problem through a loose coupling between TRIPS and OpenMath representations. Like [18], the TRIPS parser only needs to know whether a given expression is a term or a formula. We do not more tightly integrate language as our domain contains very little quantificational language of the type found in set theory (e.g., "no X is in Y"). Terms are expressions that linguistically serve as noun phrases or $\bar{N}$ constituents in sentences, for example x^3 in (1). Formulas are statements which can serve as clauses in sentences, for example, $sin'(x) = cos(x)$ in (2). Thus, we store OpenMath and TRIPS LF representations separately, passing on to TRIPS only the expression type (term or formula), and integrate the representations in interfacing to the dialogue manager.

An example representation for utterance (1) is shown in Figure 3. To be interpreted by the TRIPS parser, the input is pre-processed and the mathematical expressions are replaced with special constants which encode term types and IDs. Thus, the TRIPS parser receives as input the string *what would happen to the ˜term˜term295*. The special constant at the end indicates that it stands in for the mathematical term with ID `term295`, shown in Figure 3(b). We implemented an extension to the TRIPS tokenization algorithm which identifies these special constants and creates the corresponding LF representations in the parse, with references to the ID specified, as shown in Figure 3(a).

The example above illustrates mixing symbolic and natural language in a single utterance. In some cases, if a formula is used as part of student input, it will not be integrated with the rest of the sentence. In (1) and (2) each sentence would not be complete if the formula or term is not included. In contrast, consider the student turn $z = x^2 + 6x - 1$ *i'm not sure about what y is* In this case, the turn presents two separate contributions to dialogue: a partial solution ($z = x^2 + 6x - 1$) which by itself does not contain natural language and a separate, although related, request for help (*i'm not sure about what y is*). It is the task of the TRIPS parser to decide whether the mathematical expression is integrated with the surrounding language,

or makes a stand-alone contribution to the student turn. This decision can be made because in the latter case there is no unique parse tree covering the whole utterance. If this is the case, the parser outputs a structure labeled COMPOUND-COMMUNICATION-ACT with individual representations for different statements in the formula. The representation for *Yes, $F'(\sin x) = \cos x$* is shown here:

```
(COMPOUND-COMMUNICATION-ACT :ACTS (V11949 V11961)
   (LF::SPEECHACT V11949 W::SA_RESPONSE :CONTENT
      (W::POS :CONTENT W::YES))
   (LF::SPEECHACT V11961 W::SA_IDENTIFY :CONTENT V11945)
      (LF::THE V11945 LF::FORMULA :REFERS-TO-EXTERNAL
         W::FML_1))
```

3.2 Coverage and Error Analysis

We evaluated the accuracy of the TRIPS parser on 18 human-human tutorial dialogues we had collected in the domain of symbolic differentiation. Both student and tutor utterances were parsed, because they both contained mathematical expressions and other language students may be expected to generate. The dialogues were hand-checked using the standard TRIPS corpus methodology [17]. The accuracy results for individual dialogues are presented in Table 1. The table presents two accuracy measures: the overall accuracy measure calculated using all sentences in the dialogue, and the accuracy measure over sentences with 2 or more words. The former is a more direct reflection of expected parser performance in dialogue; but the latter is also important because one-word utterances often consist of simple acknowledgments or rejections, and excluding them represents the parser coverage on utterances where relationships between words are important.

Log #	Overall				2 or more words			
	Total	Good	Bad	Acc.	Total	Good	Bad	Acc.
1	105	64	41	61%	74	34	40	46%
2	72	53	19	74%	41	23	18	56%
3	78	49	29	63%	45	16	29	36%
4	81	54	27	67%	46	20	26	43%
5	97	57	40	59%	69	29	40	42%
6	99	69	30	70%	54	24	30	44%
7	77	63	14	82%	32	18	14	56%
8	56	39	17	70%	26	9	17	35%
9	52	26	26	50%	34	8	26	24%
10	43	22	21	51%	26	6	20	23%
11	121	76	45	63%	63	19	44	30%
12	78	57	21	73%	37	16	21	43%
13	98	50	48	51%	70	24	46	34%
14	108	77	31	71%	65	34	31	52%
15	81	51	30	63%	46	16	30	35%
16	78	52	26	67%	46	20	26	43%
17	119	89	30	75%	63	33	30	52%
18	74	48	26	65%	48	22	26	46%
19	71	39	32	55%	49	17	32	35%
Total	1588	1035	553	65%	934	388	546	42%

Table 1. Accuracy results broken by dialogue in the LeActiveMath corpus evaluation. Good, Bad - counts of utterances marked good and bad; Acc - parsing accuracy, percentage of correctly parsed utterances.

Overall, 65% of utterances in the corpus, and 42% of utterances with 2 or more words were assigned complete and correct parses by the TRIPS parser. For utterances which are ungrammatical, the parser returns a set of fragment parses, which can be ranked according to the current dialogue context by the dialogue manager, but which are not yet handled by the current version of the system.

We also conducted an error analysis to identify causes of parse failures in our corpus. Lexical failures accounted for 27% of all parse failures, and remedying the coverage gaps is a necessary goal for parser improvement. The next most common problem in the corpus, accounting for 20% of all parse failures, is the presence of spelling mistakes and abbreviations such as writing *ex* instead of *example*. We estimated the number of spelling errors in our corpus by counting the number of utterances in the corpus which contained words unknown to the parser.Overall, 7% of the utterances in the corpus contained spelling mistakes as represented by unknown words. Once these are removed, the accuracy figures for the corpus containing known words only were 71% overall and 48% for utterances with two or more words. Thus dealing with spelling mistakes and abbreviations appropriately is likely to improve system performance.

3.3 Spell-checking

Robustness is a major concern when free student input is allowed as students do not always follow the normal rules for grammar and spelling, especially in dialogue. To address misspelling we added a spelling corrector based on minimum edit distance [9, 12]. For every misspelled word, it returns a list of possible corrected spellings for words in the TRIPS lexicon. To enable the TRIPS parser to process the alternatives we used a speech lattice parsing mechanism.

The TRIPS parser can parse speech lattices which in speech applications represent uncertainty from the speech recognizer about the words recognized. This is well suited to dealing with spell-corrected output which is uncertain with respect to the appropriate correction among the possible alternative spellings . Therefore, we implemented an algorithm to convert the output of the spelling corrector into a lattice, using the number of disagreements as a score for different misspelling variants. A sample spell-corrected lattice to be sent to the TRIPS parser is shown in Figure 4.

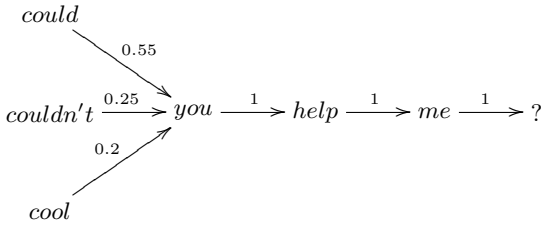

Figure 4. A lattice generated by the spelling corrector for *Coud you help me?*. Numbers on arcs indicate scores assigned to alternatives.

The TRIPS parser uses built-in word sense preferences and syntactic rules to choose the word from the lattice which best fits with the rest of the input. We evaluated the parser and spelling corrector together, with the evaluation results presented in Table 2. The table shows that spell-checking improves parsing accuracy, but that the current version of the spell-checker is not yet capable of dealing with all possible spelling mistakes, because for utterances which didn't contain typos, the parsing accuracy is significantly higher.

One reason for this difference is the weak domain model. Consider a misspelled word "oower". In absence of other information, the possible corrections are "power" or "tower", and the parser may not have sufficient evidence from the domain-independent information in the lexicon to make the right choice. Here we need a mapping which will identify words like *power* in *the power is negative* as domain concepts, and map them to corresponding concepts or symbols in the SLOPERT domain reasoner, thus excluding "tower" from consideration as it is outside the domain.

Dataset	Overall		>= 2 words	
	# of utterances	Accuracy	# of utterances	Accuracy
Raw	1588	65%	934	41%
With spell-checker	1588	66%	934	43%
Typos removed	1472	71 %	823	48%

Table 2. Evaluation summary. Raw — the original collected data; with spell-checker — raw dataset parsed with spell-checker; typos removed — accuracy on sentences with only known words.

4 Operationalizing Tutorial Dialogue Strategies

Although the information state update approach to dialogue management has been used frequently for information seeking dialogues such as searching for flights, it has less often been applied to and evaluated with intelligent tutoring systems that use deep linguistic representations. Exceptions include the BEETLE system [19] for teaching electronics which used a deep parser but not a deep generator, and the DIALOG project [3, 13] for teaching set-theoretic proofs which has not yet been evaluated.

The dialogue manager is a central component in a tutorial dialogue system and must tie together elements of domain knowledge and reasoning, pedagogical strategies, and language from other components, while still being prepared to put the current goals on hold in order to communicate with the student about basic dialogues issues such as acknowledgements, corrections, and clarification questions.

We have implemented our dialogue manager using the TRINDIKIT system [14]. The high-level generation of system feedback is performed by two key components: *UPDATE* and *SELECT* (Fig. 2). The UPDATE algorithm updates the current dialogue context (aka information state, or IS) with the dialogue moves identified by the BEET interpretation component; The SELECT algorithm then exploits the updated IS to generate high-level feedback in the form of system dialogue moves, which are then passed to the BUG generation component along with the diagnosis from the domain reasoner for verbalization.

4.1 UPDATE

The UPDATE component's task is to maintain the information state, which is the main source of combined information for all components in BEEDIFF. UPDATE's task can be described as follows:

- For each dialogue move in the sequence of moves identified by BEET, update the information state appropriately (with the student's contribution).
- For each dialogue move in the sequence of moves identified by SELECT, update the information state appropriately (with the tutor's contribution).

The dialogue manager can currently deal with three types of student moves, namely, *propose_task(SomeTask)*, *request_help()* and *give_answer(SomeAnswer)*, and three types of tutor moves, namely, *accept_task(SomeTask)*, *give_hint(SomeHint)* and *give_diagnosis(SomeDiagnosis)*.

As an example of managing the IS variables "latest_moves" and "latest_speaker", which are set by the interpretation component when the latest speaker is the student, consider the following:

```
rule( getLatestMoves,
      [ $latest_moves = M,
        $latest_speaker = DP ],
      [ set( /shared/lu/moves, M ),
        set( /shared/lu/speaker, DP ) ] ).
```

This update rule has two preconditions, which serve to retrieve the values of the IS slots "latest_moves" and "latest_speaker". In the rule's action list, this information is then stored in the "shared" part of the information state.

The next update rule example shows the interaction between a student input (student requested help) and the learner model. Note the access of the IS location "$/shared/lu/moves", which has been written by the previous rule.

```
rule( generateExerciseHelpRequestEvent,
      [ $/shared/lu/speaker == usr,
        in( $/shared/lu/moves, request_help ),
        in( $/shard/uid, UID ),
        in( $/shared/sid, SID ) ],
      [ % check Domain Reasoner for next step
        ! ( $(domain)) :: get_next_step( Step ) ] ).
```

The rule's preconditions check whether the student's latest move was indeed a help request, and retrieve both user and session ID from the information state. In the rule's action list, a query is sent to the SLOPERT domain reasoner. In addition, the update rules for student moves generate an "obligation" for the tutorial system to address the student's contribution. How the system then addresses these "obligations" is determined by SELECT, which we describe next.

4.2 SELECT

In a dialogue system, the tutor's utterances are often produced ad-hoc by a canned-text generator. Our system instead generates the tutor's output via a deep natural language generation (NLG) system, which typically consists of a set of smaller, specialized modules that produce linguistically-based textual output by iteratively refining the current dialogue move. In NLG a *dialogue planner* selects and organizes the content of the text, a *sentence planner* (or *micro-planner*) determines appropriate semantic roles for each element in the dialogue plan, and a *surface realizer* assigns syntactic roles for each semantic role, ensures each sentence is grammatical, and uses linearization (word ordering), morphology rules and lexemes to produce the final surface text. The DM's SELECT module implements the dialogue planner; the other two modules are realized in the text generation system described in Section 5.

A dialogue planner for tutorial dialogue converts tutorial strategies into turn-by-turn utterance decisions. Unlike discourse, the entire dialogue cannot be planned in advance, but must follow both an overall strategy that enforces the educational goal and a more local strategy that responds to students' answers and interruptions. In LeActiveMath, students are assumed to be working on particular problems, so there are no explicitly represented, overarching pedagogical goals, but rather smaller scoped pedagogical rules that derive directly from the domain model. We thus have both active and reactive strategies: active strategies structure the flow of the dialogue to achieve local pedagogical aims, such as teaching which derivative rule to apply in a particular situation. Reactive strategies allow for adaptive dialogue, whether by customizing the content of text to match the remedial needs of a student in a particular context, or the necessity of generating utterances whose sole purpose is to respond to obligations imposed by other participants of the dialogue (such as needing to respond to questions).

The selection and instantiation of tutorial strategies is informed by the contents of the information state, the SLOPERT domain reasoning and diagnosis engine, and the learner model. Tutorial feedback thus takes into account the context of the ongoing dialogue (in particular, the need to respond to obligations as created by UPDATE following a student contribution), the task context (as maintained by

SLOPERT), and the system's beliefs about the learner (as maintained by the learner model).

System utterances serve many pedagogical functions, including feedback for previous student answers, cues to elicit new information, explanations to eliminate student misconceptions, or even motivation to keep students on track. They are also important for redirecting the dialogue via backoff strategies when the parser fails, although as of now we have not yet implemented such strategies.

The dialogue planner must then map these diverse intentions to a sequence of dialogue moves (type and propositional content) whose surface text realizes these goals.

In the initial version of SELECT, we have implemented three simple feedback strategies:

- Accept a task whenever the learner proposes one.
- Give a hint whenever help is requested, guiding the learner through a solution graph.
- Give a diagnosis of input whenever the learner proposes a term as a solution or intermediary step.

An encoding of the latter two types of SELECT rules is given below:

```
rule( giveHint,
      [ $/shared/lu/speaker == usr,
        in($/shared/lu/moves, request_help) ],
      [ % check SLOPERT for specific hint
        ! ($(domain)) :: get_next_step( Step ),
        % make the tutor give the hint
        add(next_moves, give_hint( Step )) ] ).

rule( giveDiagnosis,
      [ $/shared/lu/speaker == usr,
        in($/shared/lu/moves,
           give_answer(StudentAnswer)) ],
      [ % check availability of diagnosis
        ! ($(domain)) ::
           get_diagnosis(StudentAnswer, Diagnosis),
        add(next_moves,
            give_diagnosis(Diagnosis)) ] ).
```

When a hint is needed, SLOPERT produces a set of possible hints which the SELECT module has the responsibility to convert into a context-sensitive hint. Note that the specificity of the hint can be influenced by the contents of the learner/situational model. If the learner's general aptitude or prior performance, for instance, is "good", then the SELECT module may generate a hint of low specificity, say by giving the name of the next differentiation rule to apply. If the system judges the learner as "weak", then hints could be more informative, say, by telling the learner the name of the rule as well as its symbolic representation.

Note also that both rules add dialogue moves to the IS slot "next_moves". Upon activation by TRINDIKIT, the BUG verbalisation component retrieves the value of this IS slot, processes it to produce the text for the dialogue moves, and writes the result of verbalisation to the IS slot where it can be read by the G(LE)^{2}AMuser interface to show to the student.

5 Generating Tutorial Utterances

We have implemented an initial version of the text generation component (named BUG) that can produce utterances for the computer in its role as tutor. BUG can generate a variety of verbalizations for system feedback within the current context and automatically revise small chunks of these written dialogue segments into connected, natural prose. Tasked with producing tutor utterances, the generation system must correctly handle linguistic phenomena both at and above the sentence level and that originates on the part of either the student or tutor. These linguistic phenomena are independent of domain (mathematics, flight reservations, systems control, etc.) and include correctly pronominalizing concrete references in a natural way, how to appropriately combine multiple dialogue moves (sentences) into a single turn, and deciding which elements can be elided.

5.1 Linguistic Phenomena in Dialogue: Pronominalization, Revision and Ellipsis

Like discourse generation [5], creating text for use in a dialog requires linguistic rules for correctly generating pronouns, revising small sentences into larger sentences, lexically marking relations between sentences, and generating ellipsis.

Pronouns: Dialogue situations require some modification to rules that have been previously applied in discourse-only applications. For instance, tutors refer to the student as "you", both participants as "we", and students refer to themselves as "I":

Tutor: In that last one did you mean dx/dw?
Tutor: Last week we solved $sin(6x-2)^3$.

Other discourse elements are typically marked with neuter gender:

Student: The answer is $18cos(sin(6x-2))^2$.
Tutor: Almost, try to tidy it up.

This means that both dialogue participants and content must be explicitly represented in each utterance to be correctly realized.

We have adapted a previous, discourse-based pronominalization solution [6] to serve as the pronominalization module in LeActiveMath. The principal difference is the generation component is only responsible for tutor utterances, and thus when the student produces referring expressions during their turn, the interpreter component must pass these along (via the information state) to the generation component. We thus had to provide a mechanism to communicate shared ID's between the interpreter and generator via the information state. For example:

Student: Did I leave out the minus sign?
References: "I": student001, "minus sign": minus-sign001

Tutor: Yes, it should be there.
References: "it": minus-sign001

In this case, without knowing that the student had just mentioned the concept "minus sign", the generation component would not have known it should make it a pronoun.

This process must also work in reverse; namely, the generation system must inform the interpretation component which concrete references have been produced for the tutor's utterance, so that it may update its dialogue context and be more likely to correctly resolve pronouns uttered by the student. Again, the information state serves as a container or blackboard for the interpretation and generation components. This requires both a common lexicon and common set of mentionable entities in the knowledge representation.

Revision: Another dialogue-level linguistic phenomena required for high-quality text is that of *revision*, which can consist of either *text plan rearrangement* or *clause aggregation*. Revision is important not only because it makes the text more readable, but it has also been shown to improve learning gains compared to non-revised text [11]. In general, SELECT will generate a sequence of dialogue moves to BUG for verbalisation within a single turn. For instance, here a set of three dialogue moves should be aggregated for readability, as the collected corpus has shown:

Tutor: Ok. [SignalUnderstanding]
Tutor: Exactly. [Correctness]
Tutor: Go on to the next step. [ActionDirective]

Tutor: Ok, exactly, now go on to the next step.

At other times, an explicit discourse relation might exist between multiple moves, as with "but" in this example:

Tutor: You got the answer mostly right. [Partial-Correctness]
Tutor: [Incompleteness-Relation]
Tutor: You didn't tidy it up. [Assert]

Tutor: You got the answer mostly right, but you didn't tidy it up.

Ellipsis: Finally, the language of dialogue, and our corpus in particular, is full of syntactically incomplete utterances where missing elements can be filled in by previous elements of the dialogue [4, 10]. For instance:

Student: Should the minus signs be on the inside or the outside?
Tutor: <*The minus signs should be*> On the outside.

requires that the generator recognize that overlap in syntactic forms is occurring and that only the newest information need be presented. Like pronominalization, this requires a degree of communication with the interpreter. Although we have not fully specified the format of this communication, it should be necessary to include high-level (e.g., chunked) syntax. We have however tested the generation system on such examples to ensure that it can produce appropriate ellipsis when the appropriate mechanism is fully defined.

5.2 Generation System Architecture

Rather than create an entire text generation system from scratch, we have modified the STORYBOOK system [5] originally intended for discourse to generate tutorial utterances for dialogue. STORYBOOK takes speech-act-like representations and brings to bear a lexicon, grammar, and rules for determining pronouns, clause aggregations and discourse markers to produce fluent text. BUG retrieves dialogue moves from the IS and also accesses other parts of the dialogue context (say, for references or past student performance). Examples of simple utterances and the dialogue moves for producing them are:

Let's differentiate <expression1>
`accept_task(diff(<expression1>))`
If you apply <rule> to <expr1>, you obtain <expr2>.
`give_hint(apply_rule(<rule>, <expr1>, <expr2>))`
You should identify the form of the statement.
`give_hint(identify_form(<expression>))`
You need to perform a substitution.
`give_hint(substitute(<input>, <subst>, <output>))`
You missed the inner layer.
`give_diagnosis(applied_rule(buggy,<rule>,<expr1>, <expr2>,error(missing_inner_layer)))`

Unlike the interpreter, we have a great degree of control over the language we would like to produce for the system's tutorial utterances, and thus we do not need as many new vocabulary items or grammar rules. We can also ensure the coverage of the utterance generator by testing that it could successfully generate all utterances in several of the dialogues from our corpus. The following are examples of typical tutor utterances from our human-human corpus that the system is capable of producing:

Tutor: Try $sin(x^4 - 2x)$.
Tutor: Can you tidy up the minus signs?
Tutor: Try $cos(7 - x^3)$ where cos differentiates to -sin.

The final result is a text string that represents an appropriate system tutorial utterance that corresponds to the dialogue obligations and content as specified by SELECT, with optional multi-modal effects for the G(LE)^{2}AM graphical user interface, such as red highlights around sections of algebraic expressions currently under the focus of tutor and student. The generated string is stored in the IS where G(LE)^{2}AMcan retrieve it to display to the student.

6 An Example Interaction

In this section, we briefly describe the flow of control between the various DM components for typical dialogue interactions.

A learner is first engaged in exploring lessons on differentiation with the LeActiveMath system, and at some point clicks on a link that has been generated by the Tutorial Component for her. This link points to an interactive exercise on symbolic differentiation with support for natural-language enhanced tutorial dialogue. The system is currently set to allow the student to propose problems using the G(LE)^{2}AM editor rather than the LeActiveMath system, although in the future either will be able to do so.

Suppose the learner first enters the phrase "Let's differentiate" and composes the math term $sin(x^2)$, then clicks on the "Submit" button shown in Figure 1. The term is first displayed in the Previous Term window (for subsequent Copy&Paste) and the dialogue history window is updated with her contribution. The applet then sends her input to the dialogue manager, which causes UPDATE to augment the information state with:

`latest_moves=propose_task(diff(`$sin(x^2)$`))` and
`latest_speaker=learner`

SELECT then computes the tutorial system's feedback by finding a rule that can accept the new problem definition and checking with the user model to determine that the problem is not too difficult:

`next_moves=accept_task(diff(`$sin(x^2)$`))`

UPDATE again augments the information state with the tutor's dialogue move, the SLOPERT domain reasoner initializes itself for the new differentiation task, and BUG is activated, whereupon it retrieves the dialogue move from the IS and verbalizes it into English, say with the following string that is itself added to the IS:

Okay, let's differentiate $sin(x^2)$.

UPDATE then adds this string to the IS's `output` slot, copies the value of the `next_moves` slot into `latest_moves`, and sets the `latest_speaker` slot to `tutor`. The string is now sent to the G(LE)^{2}AM applet, where it is added to the dialogue history window, allowing the learner to read it. The "Current Problem" window is then updated with the (now proposed and accepted) task.

The applet then waits for the student to begin solving the problem. When the student composes a math term, say $cos(x^2)$, and clicks the "Submit" button, BEET is called to parse the learner's input. In this case, it only contains a term, and BEET thus updates the IS with:

`latest_moves=give_answer(`$cos(x^2)$`)` and
`latest_speaker=learner`

In the dialogue manager, a SELECT rule fires which generates the system's intention to diagnose the student's answer. The IS is thus updated with:

`next_moves=give_diagnosis(SomeDiagnosis)`

where SomeDiagnosis is being computed by SLOPERT, in this case as MISSING_INNER_LAYER. BUG is now activated, takes the dialogue move(s) representing the tutorial utterance, and produces the utterance for the new diagnosis, *e.g.: "You missed the inner layer"*. The process above continues, where the IS variables `latest_moves`, `latest_speaker`, etc. are constantly updated as the dialogue turn is exchanged between tutor and student.

Supposing the student types "I need help, please", BEET is again called to parse the learner's English input. A successful parse will update the IS with:

`latest_moves=help_request` and
`latest_speaker=learner`

A SELECT rule fires which generates the system's intention to address the learner's help request with a hint, causing the IS to be updated with:

`next_moves=give_hint(SomeHint)`

where SomeHint is computed by a call to the domain reasoner (since it has a representation of the problem-solving state and the appropriate next step). BUG is then activated again, and generates a textual hint based on SLOPERT's high-level conceptual hint, such as "Apply the chain rule." This process continues until the learner eventually arrives at the correct answer, whereupon the tutorial system congratulates her and she is allowed to return to the LeActiveMath main system or else to continue solving differentiation problems.

7 Evaluation Plans and Future Work

The BEEDIFF system will be a platform for evaluating many aspects of tutoring mathematics with natural language dialogue. Our first evaluation will focus on whether students learn more when given a chance to interact via natural language dialogue compared with a button-based GUI that allows highly restricted interaction. We expect that a dialogue-style interaction allowing students to use written text to hold an in-domain dialogue will improve learning gain. Such a study would necessitate a great degree of robustness, which forces us to also focus on engineering issues such as spelling correction, as a system which could handle only a fraction of student input would not be a fair evaluation of the language hypothesis.

A second evaluation would center around the effects of adaptivity of the generated tutor utterances on student performance. This would require a working system with two versions of the BUG component: one that is not adaptive and produces identical utterances given identical tutoring states, and a fully adaptive version that also looks into for instance the student model, recent errors and misconceptions, and number of hints needed to date to produce tailored utterances that would be more helpful in a given situation. Finally, we are also interested in dynamically adjusting the pedagogical strategy to see if we can provide evidence for various tutoring theories.

8 Conclusions

We have described the design and preliminary implementation of several components for natural-language enhanced tutorial dialogue in the tutoring domain of symbolic differentiation. We have implemented initial versions for interpretation (BEET), information state update (UPDATE), feedback selection (SELECT), and generation (BUG), plus the SLOPERT domain reasoner. The input and output components have been realised as the G(LE)^{2}AMgraphical user interface. As a whole, the system allows learners to conduct basic dialogues while solving differential equations where the tutoring system can distinguish correct from incorrect answers, provide diagnoses when it detects common misconceptions, and respond intelligently to help requests. We expect to continue to develop the system to achieve human-level tutorial proficiency for differential equations.

REFERENCES

[1] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, 'Cognitive tutors: Lessons learned', *The Journal of the Learning Sciences*, **4**(2), 167–207, (1995).

[2] B. S. Bloom, 'The two sigma problem: The search for methods of group instruction as effective as one-to-one tutoring', *Educational Researcher*, **13**, 3–16, (1984).

[3] Mark Buckley and Christoph Benzmueller, 'System description: A dialogue manager supporting natural language tutorial dialogue on proofs', in *Proceedings of the ETAPS Satellite Workshop on User Interfaces for Theorem Provers*, pp. 40–67, Edinburgh, Scotland, (2005).

[4] Charles Callaway, 'Do we need deep generation of disfluent dialogue?', in *Proceedings of the AAAI Spring Symposium on Natural Language Generation in Spoken and Written Dialogue*, pp. 6–11, Palo Alto, CA, (March 2003).

[5] Charles B. Callaway and James C. Lester, 'Narrative prose generation', *Artificial Intelligence*, **139**(2), 213–252, (August 2002).

[6] Charles B. Callaway and James C. Lester, 'Pronominalization in generated discourse and dialogue', in *Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 88–95, Philadelphia, PA, (July 2002).

[7] Myrosia Dzikovska, David Reitter, Johanna D. Moore, and Claus Zinn, 'Data-driven modeling of human tutoring in calculus', in *Proceedings of the Workshop on Language-Enhanced Educational Issue*, Riva del Garda, Italy, (2006). in this issue.

[8] Myroslava Dzikovska, Mary Swift, James Allen, and William de Beaumont, 'Generic parsing for multi-domain semantic interpretation', in *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT-05)*, Vancouver, (October 2005).

[9] M. Elmi, *A Natural Language Parser with Interleaved Spelling Correction*, Ph.D. dissertation, Illinois Institute of Technology, 1994.

[10] Stina Ericsson, 'A corpus study towards the generation of elliptical utterances in a dialogue system', in *Proceedings of the ESSLLI Workshop on Cross-modular Approaches to Ellipsis*, Edinburgh, UK, (August 2005).

[11] B. Di Eugenio, D. Fossati, D. Yu, S. Haller, and M. Glass, 'Natural language generation for intelligent tutoring systems: A case study', in *Proceedings of the 12th International Conference on Artificial Intelligence in Education*, Amsterdam, The Netherlands, (July 2005).

[12] Martha Evens and Mohammad Ali Elmi. Spelling correction using context*, April 04 2002.

[13] Helmut Horacek and Magdalena Wolska, 'A hybrid model for tutorial dialogs', in *Proceedings of the 6th SIGDial Workshop on Discourse and Dialogue*, pp. 190–199, Lisbon, Portugal, (2005).

[14] Staffan Larsson and David Traum, 'Information state and dialogue management in the TRINDI dialogue move engine toolkit', *Natural Language Engineering*, **6**(3–4), 323–340, (2000).

[15] N. Person, A.C. Graesser, D. Harter, and E. Mathews, 'Dialog move generation and conversation management in autotutor', in *Workshop Notes of the AAAI '00 Fall Symposium on Building Dialogue Systems for Tutorial Applications*, pp. 45–51, Cape Cod, MA, (November 2000).

[16] Carolyn Rosé, Pamela Jordan, Michael Ringenberg, Stephanie Siler, Kurt VanLehn, and Anders Weinstein, 'Interactive conceptual tutoring in atlas-andes', in *Proceedings of AI in Education 2001 Conference*, (2001).

[17] Joel Tetreault, Mary Swift, Preethum Prithviraj, Myroslava Dzikovska, and James Allen, 'Discourse annotation in the monroe corpus', in *ACL workshop on Discourse Annotation*, Barcelona, Spain, (July 2004).

[18] Magdalena Wolska and Ivana Kruijff-Korbayová, 'Analysis of mixed natural and symbolic lanugage input in mathematical dialogs', in *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pp. 25–32, Barcelona, Spain, (July 2004).

[19] Claus Zinn, Johanna Moore, and Mark Core, 'Intelligent information presentation for tutoring systems', in *Multimodal Intelligent Information Presentation*, 227–252, Springer, Dordrecht, The Netherlands, (2005).